Eye Controlled WheelChair

Gourav Kumar 17115037
Himani Jain 17115040
Rahul Badwaik 17115069

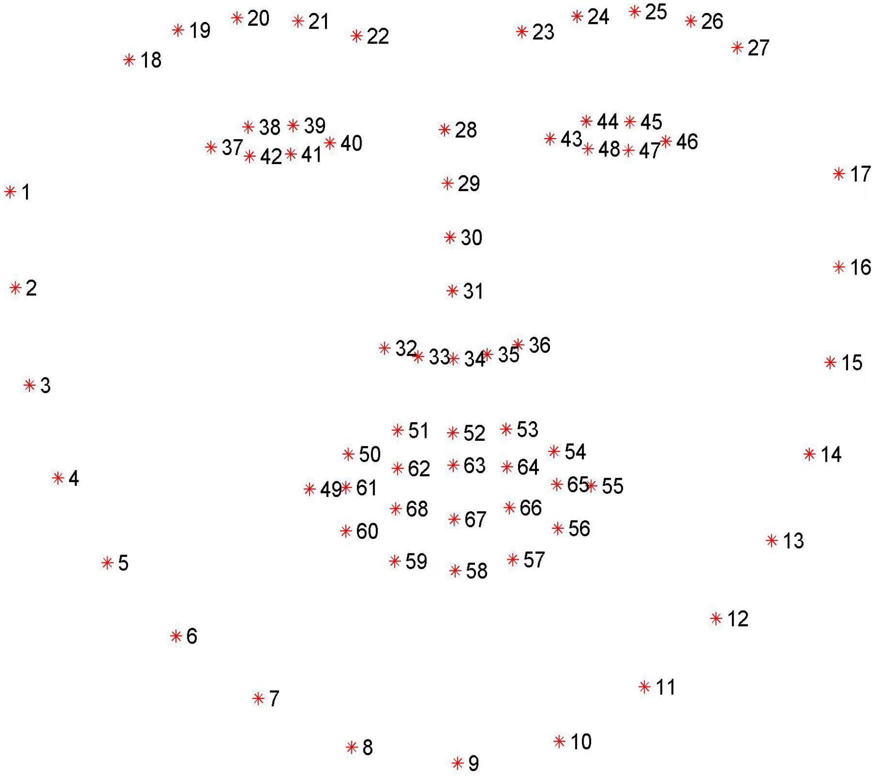
Problem Statement

Locomotion is important for keeping up with the pace of life. There are many people who are unable to move from one place to another on their own because of physical disability.

People who have lost muscle control cannot operate wheelchair on their own. Traditional wheelchairs are difficult to operate by self. An individual might be partially paralyzed hence would always require external help.

Our project aims at making life of these people easier and self reliant. The idea is to control the movement of wheelchair with the motion of pupil.

Facial Landmarks using DLIB predictor and OpenCV

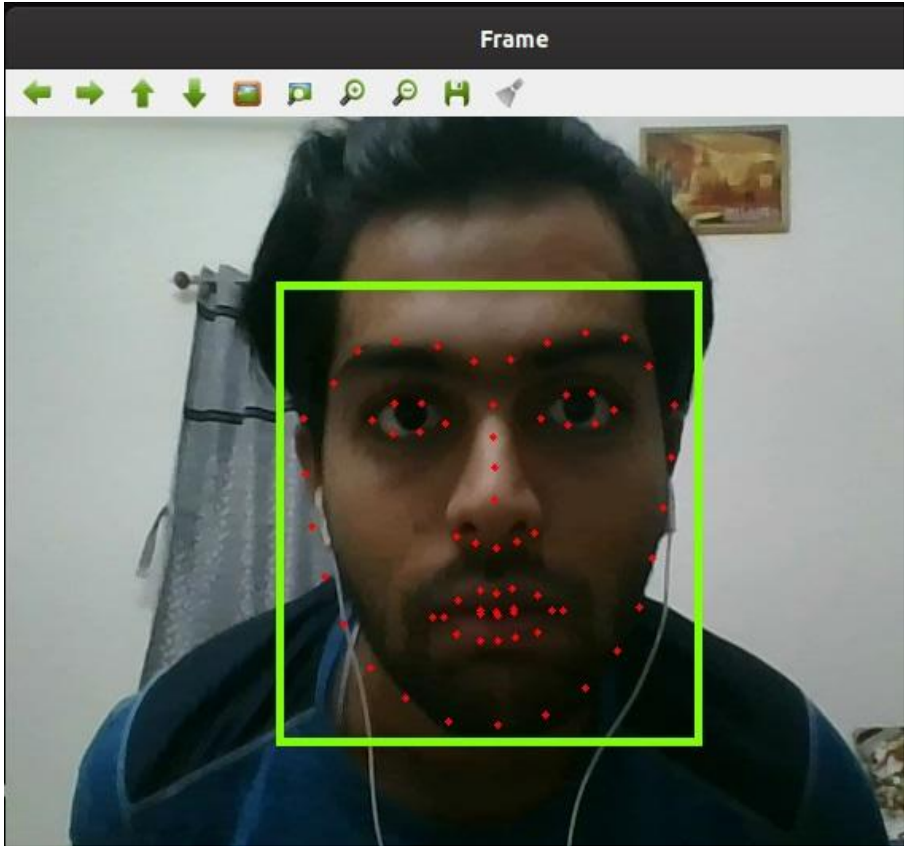


Detecting facial landmarks is a subset of the shape prediction problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In the context of facial landmarks, goal is to detect important facial structures on the face using shape prediction methods.

Points for detection of eyes:

- Left Eye : (37, 38, 39, 40, 41, 42)
- Right Eye : (43, 44, 45, 46, 47, 48)



Detecting facial landmarks is therefore a two step process:

- Localize the face in the image.
- Detect the key facial structures on the face ROI.

Facial landmarks are used to localize and represent salient regions of the face, such as:

Eyes
Eyebrows
Nose
Mouth
Jawline

Facial landmarks have been successfully applied to face alignment, head pose estimation, face swapping, blink detection and much more.

Function to determine the extent to which the eye is open

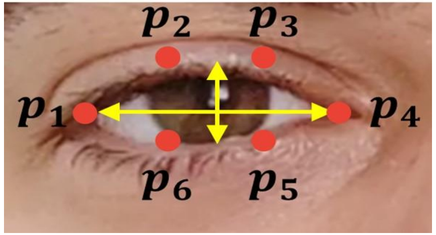
```python
def get_blinking_ratio(eye_points, facial_landmarks):
    left_point = (facial_landmarks.part(eye_points[0]).x, facial_landmarks.part(eye_points[0]).y)
    right_point = (facial_landmarks.part(eye_points[3]).x, facial_landmarks.part(eye_points[3]).y)
    center_top = midpoint(facial_landmarks.part(eye_points[1]), facial_landmarks.part(eye_points[2]))
    center_bottom = midpoint(facial_landmarks.part(eye_points[5]), facial_landmarks.part(eye_points[4]))

    #hor_line = cv2.line(frame, left_point, right_point, (0, 255, 0), 2)
    #ver_line = cv2.line(frame, center_top, center_bottom, (0, 255, 0), 2)

    hor_line_lenght = hypot((left_point[0] - right_point[0]), (left_point[1] - right_point[1]))
    ver_line_lenght = hypot((center_top[0] - center_bottom[0]), (center_top[1] - center_bottom[1]))

    ratio = hor_line_lenght / ver_line_lenght
    return ratio
```

$$\text{Blinking ratio} = \frac{\text{Horizontal length}}{\text{Vertical length}}$$



Function to determine the direction of gaze using left/right gaze ratio

```
def get_gaze_ratio(eye_points, facial_landmarks, frame, gray):
    left_eye_region = np.array([(facial_landmarks.part(eye_points[0]).x, facial_landmarks.part(eye_points[0]).y),
                                (facial_landmarks.part(eye_points[1]).x,
                                facial_landmarks.part(eye_points[1]).y),(facial_landmarks.part(eye_points[2]).x,
                                facial_landmarks.part(eye_points[2]).y),(facial_landmarks.part(eye_points[3]).x,
                                facial_landmarks.part(eye_points[3]).y),(facial_landmarks.part(eye_points[4]).x,
                                facial_landmarks.part(eye_points[4]).y),(facial_landmarks.part(eye_points[5]).x,
                                facial_landmarks.part(eye_points[5]).y)], np.int32)
    # cv2.polylines(frame, [left_eye_region], True, (0, 0, 255), 2)
    height, width, _ = frame.shape
    mask = np.zeros((height, width), np.uint8)
    cv2.polylines(mask, [left_eye_region], True, 255, 2)
    cv2.fillPoly(mask, [left_eye_region], 255)
    eye = cv2.bitwise_and(gray, gray, mask=mask)
    min_x = np.min(left_eye_region[:, 0])
    max_x = np.max(left_eye_region[:, 0])
    min_y = np.min(left_eye_region[:, 1])
    max_y = np.max(left_eye_region[:, 1])
    gray_eye = eye[min_y: max_y, min_x: max_x]
    _, threshold_eye = cv2.threshold(gray_eye, 70, 255, cv2.THRESH_BINARY)
    height, width = threshold_eye.shape
    left_side_threshold = threshold_eye[0: height, 0: int(width / 2)]
    left_side_white = cv2.countNonZero(left_side_threshold)
    right_side_threshold = threshold_eye[0: height, int(width / 2): width]
    right_side_white = cv2.countNonZero(right_side_threshold)

    if left_side_white == 0:
        gaze_ratio = 1
    elif right_side_white == 0:
        gaze_ratio = 5
    else:
        gaze_ratio = left_side_white / right_side_white

    return gaze_ratio
```

Main function to run in infinite loop:

```
if blinking_ratio > 5.7:
    if count==3:
        a=0
        print("START")
    if count==5:
        a=4
        count=0
        print("END")
    count=count+1
else:
    if gaze_ratio <= 0.8:
        cv2.putText(frame, "RIGHT", (50, 100), font, 2, (0, 0, 255), 3)
        new_frame[:] = (0, 0, 255)
        a=1
        print("RIGHT")
        print(gaze_ratio)
    elif 2 < gaze_ratio <= 4:
        cv2.putText(frame, "CENTER", (50, 100), font, 2, (0, 0, 255), 3)
        a=2
        print("CENTRE")
        print(gaze_ratio)
    else:
        new_frame[:] = (255, 0, 0)
        cv2.putText(frame, "LEFT", (50, 100), font, 2, (0, 0, 255), 3)
        a=3
        print("LEFT")
        print(gaze_ratio)
```

Result

The system works with an accuracy rate of 70-90 % which is quite satisfactory. The image capture, eye movement detection and the algorithm for validating movement attempts perform reliably.

The wheelchair follows the following output

Conclusion

Quadriplegia is paralysis caused by illness or injury to the humans that results in partial or complete loss of limbs and torso. It's a phenomenon which confines the ability of a person to move by himself, and he has to rely on someone to carry him around.

We wanted to utilize the opportunity to design something which could be a contribution in our own small way to the society.

Following limitations need to be addressed:

- User’s face should be well illuminated.
- Head of the person should be steady for precise detection.
- Camera should be at a specific distance from user and background should not have any other faces.

Future Scope

- Cursor Tracking for PCs
- Completely paralyzed person can use his pupil motion for indication ‘YES?NO’.

THANK YOU